MODULE 4

Session 3 of 4

Configuration, Option & Structured Logging

TMS secrets, typed settings, searchable logs, correlation

Session 3 Production-Ready Configuration & Logs

By the end of this session, you will:

01 Protect secrets

No passwords in source or committed JSON env vars / vault / user-secrets

02 Options pattern

Typed, bound configuration not stringly-typed IConfiguration lookups

03 Fail at startup

ValidateOnStart missing config before users, not hours after deploy

04 Structured logging

Named properties aggregators can index not concatenated prose

05 Log levels

Critical → Trace pick the right noise level for TMS in prod vs debug

06 Correlation IDs

One ID per request follow an enrollment through every log line

Where Do Database Passwords Live?

Reject in code review

```
// NEVER  hardcoded

var conn = "Server=prod;Password=...";
// NEVER  secrets in Git

// appsettings.json with real password
```

Correct: environment variables, Key Vault, or (dev only) dotnet user-secrets not version control.

Configuration hierarchy (last wins)

1 appsettings.json defaults (committed)

2 appsettings.{Environment}.json

3 User Secrets dev machine only

4 Environment variables deploy-time

5 Command-line args highest priority

TMS rule of thumb

appsettings: timeouts, feature flags, log levels non-secrets. Connection strings with secrets: override from env or vault per environment.

The Options Pattern Typed Configuration

String lookups

```
// Fragile  string keys

var port = int.Parse(_config["Server:Port"] ?? "5000");
// Typo in key? Runtime crash.
```

DatabaseOptions.cs

```
public class DatabaseOptions
{
  public const string SectionName = "Database";
  public required string ConnectionString { get; init; }
  public int Timeout { get; init; } = 30;
}
```

Program.cs fail at startup

```
builder.Services
  .AddOptions<
DatabaseOptions>()

.BindConfiguration(DatabaseOptions.SectionName)

.ValidateDataAnnotations()

.ValidateOnStart();
```

```
{ "Database": { "ConnectionString": "...", "Timeout": 30 } }
```

Why teams adopt Options

Typed access, validation, and startup checks remove ConnectionString from appsettings with ValidateOnStart and the process refuses to start instead of failing on first enrollment.

Options Validation Fail Before Users

DataAnnotations

```
public class DatabaseOptions
{
  [Required]
  public required string ConnectionString { get; init; }
  [Range(1, 300)] public int Timeout { get; init; }
}
```

Why is startup failure better than failing on first use?

If ConnectionString is missing at startup:

```
Microsoft.Extensions.Options.
OptionsValidationException:
  DataAnnotation validation failed for
  'DatabaseOptions':
  'Database connection string is required'

→ App REFUSES to start.
```

Ops perspective

Bad: app runs three hours, looks healthy, first enrollment crashes team thought deploy succeeded. Good: pipeline or container restart loops on a clear validation error no user traffic yet.

Injecting Options into TMS Services

Primary constructor (C# 14)

```
public class EnrollmentService(IOptions<DatabaseOptions> db,
    ILogger<EnrollmentService> log)
{
  private readonly DatabaseOptions _db = db.Value;
  // TimeSpan.FromSeconds(_db.Timeout) ...
}
```

For TMS in this course: IOptions<T> is enough config changes mean redeploy.

`IOptions<T>`	**Snapshot at startup** TMS default restart picks up changes
`IOptionsSnapshot<T>`	**Per-request read** When config can change without restart
`IOptionsMonitor<T>`	**Change notifications** React when settings reload

Structured log line in the same method

logger.LogInformation("Processing enrollment for {StudentId} in {CourseCode}", studentId, courseCode);

Structured Logging Finding the Needle

A million lines/day enrollment 12345 failed. Plain text search returns noise (ports, timestamps, partial IDs).

Not indexed as fields

```
// BAD  concatenation / pre-formatted

logger.LogInformation("User " + name + ...");

logger.LogInformation($"User {name} ..."); // $ loses
separate properties
```

Seq / App Insights / ES

```
// GOOD  template + values

logger.LogInformation(
  "User {StudentName} enrolled in {CourseCode}",
  name, course);
```

Query example in an aggregator:

```
StudentName = "Abeba" AND CourseCode = "CS101"
```

Curly braces in the template are property names not the same as string.Format positions only.

Remember

Braces name fields for indexes. $"..." collapses everything into one string you lose structured search.

Log Levels Signal vs Noise

Critical	**App must stop**	TMS: DB permanently unavailable
Error	**Operation failed**	TMS: Enrollment rejected
Warning	**Suspicious, recovering**	TMS: Retry 2 of 3 on outbound call
Information	**Notable business event**	TMS: Enrollment processed
Debug	**Developer flow detail**	TMS: EF SQL text, cache miss
Trace	**Per-line verbosity**	TMS: Method entry/exit

Production-style minimums (raise per-namespace temporarily when debugging):

```
{
"Logging": {
  "LogLevel": {
    "Default": "Information",
    "Microsoft.AspNetCore": "Warning",
    "Microsoft.EntityFrameworkCore": "Warning"
  }
}
}
```

Correlation IDs One Request, One Story

Middleware resolve ILogger as today’s Session 1 stack allows

```
app.Use(async (context, next) =>
{
  var id = context.Request.Headers
    ["X-Correlation-ID"].FirstOrDefault()
    ?? Guid.NewGuid().ToString();
  context.Response.Headers
    ["X-Correlation-ID"] = id;
  using (logger.BeginScope(
    new Dictionary<string,object>
      { ["CorrelationId"] = id }))
    await next(context);
});
```

Support workflow

· Student: “Enrollment failed.”

· Support: paste CorrelationId from UI / response header.

· Search: CorrelationId = "abc-123"

· → full chain: validate → capacity → error

```
[Info] CorrelationId=abc-123 | Processing enrollment STU-001

[Info] CorrelationId=abc-123 | Validating course capacity

[Error] CorrelationId=abc-123 | Enrollment failed: capacity reached
```

Without correlation

Millions of interleaved lines you guess which Information entries belong together. With BeginScope + ID, aggregators filter one thread of work.

Configuration & Logging Real-World Mistakes

Same theme as Sessions 1–2: looks fine in demo, fails or blinds you in production.

Scenario 1 secrets in repo

Git history never forgets

```
// appsettings.Development.json  committed

"ConnectionStrings": {

"TMS": "Host=...;Password=real"
}
```

Result: anyone with clone access has production credentials; scanners and forks multiply exposure.

Scenario 2 “structured” logging that is not

EnrollmentService

```
logger.LogInformation(
  $"Enrolled {studentId} in {course}"); // one string blob
```

Result: Seq/App Insights cannot index StudentId / CourseCode as separate fields you are back to substring search and false positives.

Contrast with slides 3–7

Secrets via env/vault + user-secrets in dev; Options + ValidateOnStart; template logging with explicit property names; correlation scope for one request’s trail.

Lab 3

Options, Logging & TMS

Exercises 3, 4, and 5 workbook is source of truth

1. M4-Session-3-Presentation.md Lab handoff Session 3
2. M4-Lab-Workbook.md Exercise 3, Exercise 4, Exercise 5 (full steps)
3. Confirm options + logging checkpoint before Session 4 capstone prep

This deck stays concept-first execution lives in labs deck + workbook

Session 3 Summary What You Learned

Configuration hierarchy	Later sources override earlier env beats files; never commit secrets.
Options pattern	Typed sections, BindConfiguration, annotations compile-time shape for TMS settings.
ValidateOnStart()	Invalid or missing config fails process start not first user action.
Structured logging	Template + named holes → indexable StudentId, CourseCode, EnrollmentId.
Log levels	Information baseline in prod; dial Debug per namespace briefly when investigating.
Correlation IDs	Scope + header one filter shows the whole enrollment attempt.

Reflection Fail Fast, Search Easy

Take 4 minutes solo notes or pair:

· Where would you put the TMS production connection string and what never goes in Git?

· Why is ValidateOnStart better than discovering bad config on the first enrollment?

· How would correlation IDs change how you debug a support ticket?

Facilitator: connect to Options slide remove Database:ConnectionString locally, run show the startup exception in the terminal so the room hears the “fail loud” story.

What This Looks Like in an Interview

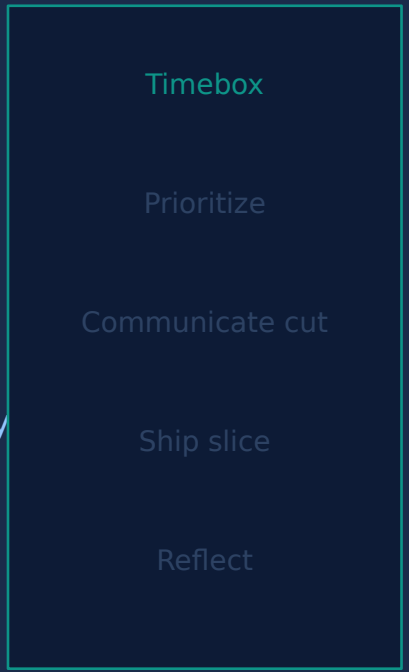
Configuration management is often overlooked by junior developers. Companies notice those who validate at startup rather than discovering missing settings in production at 2am. Structured logging with correlation IDs is what separates someone who "writes code" from someone who "runs systems."

Bring one TMS example: named properties on an enrollment log line + where the correlation ID appears on the API response tie story to observability you would actually run.

Coming Up in Session 4

Capstone Pressure Rehearsal

Working API, three change requirements at time zero, ~30 minutes goal is decision-making under pressure: what do you ship first, what do you consciously defer? Same trade-offs professional teams make daily.



Not a gate you can fail a rehearsal